Forgetting under the Well-Founded Semantics

José Júlio Alferes\textsuperscript{1}, Matthias Knorr\textsuperscript{1}, and Kewen Wang\textsuperscript{2}

\textsuperscript{1} CENTRIA & Departamento de Informática, Faculdade Ciências e Tecnologia, Universidade Nova de Lisboa, 2829-516 Caparica, Portugal
\textsuperscript{2} School of Information and Communication Technology, Griffith University, Brisbane QLD 4111, Australia

Abstract. In this paper, we develop a notion of forgetting for normal logic programs under the well-founded semantics. We show that a number of desirable properties are satisfied by our approach. Three different algorithms are presented that maintain the computational complexity of the well-founded semantics, while partly keeping its syntactical structure.

1 Introduction

Forgetting has drawn considerable attention in knowledge representation and reasoning. This is witnessed by the fact that forgetting has been introduced in many monotonic and nonmonotonic logics \cite{1,5,9,10,11,12,16,18,19}, and in particular, in logic programming \cite{6,15,17}.

A potential drawback, common to all these three approaches, is the computational (data) complexity of the answer set semantics, which is \textsc{coNP}, while the other common semantics for logic programs, the well-founded semantics (WFS), is in \textsc{P}, which may be preferable in applications with huge amounts of data. However, to the best of our knowledge, forgetting under the well-founded semantics has not been considered so far. Therefore, in this paper, we develop a notion of forgetting for normal logic programs under the well-founded semantics. We show that forgetting under the well-founded semantics satisfies the properties in \cite{6}. In particular, our approach approximates semantic forgetting of \cite{6} for normal logic programs under answer set semantics as well as forgetting in classical logic, in the sense that whatever is derivable from a logic program under the well-founded semantics after applying our notion of forgetting, is also derivable in each answer set and classical model after applying semantic and classical forgetting to the logic program and its classical representation, respectively. We also present three different algorithms that maintain the favorable computational complexity of the well-founded semantics when compared to computing answer sets.

2 Preliminaries

A normal logic program $P$, or simply logic program, is a finite set of rules $r$ of the form $h \leftarrow a_1, \ldots, a_n, \neg b_1, \ldots, \neg b_m$ where $h, a_i$, and $b_j$, with $1 \leq i \leq n$ and $1 \leq j \leq m$, are all propositional atoms over a given alphabet $\Sigma$. 

Given a rule \( r \), we distinguish the head of \( r \) as \( \text{head}(r) = h \), and the body of \( r \), \( \text{body}(r) = \text{body}^+(r) \cup \text{not body}^-(r) \), where \( \text{body}^+(r) = \{ a_1, \ldots, a_n \} \), \( \text{body}^-(r) = \{ b_1, \ldots, b_m \} \) and, for a set \( S \) of atoms, \( \text{not } S = \{ \text{not } q \mid q \in S \} \). Rule \( r \) is positive if \( \text{body}^-(r) = \emptyset \), negative if \( \text{body}^+(r) = \emptyset \), and a fact if \( \text{body}(r) = \emptyset \).

Given a logic program \( P \), \( B_P \) is the set of all atoms appearing in \( P \), and \( \text{Lit}_P = B_P \cup \text{not } B_P \). Also, \( \text{heads}(P) \) denotes the set \( \{ p \mid p = \text{head}(r) \land r \in P \} \).

A three-valued interpretation \( I = I^+ \cup \text{not } I^- \) with \( I^+, I^- \subseteq B_P \) and \( I^+ \cap I^- = \emptyset \). Informally, \( I^+ \) and \( I^- \) contain the atoms that are true and false in \( I \), respectively. Any atom appearing neither in \( I^+ \) nor in \( I^- \) is undefined.

We recall the definition of the well-founded semantics based on the alternating fixpoint [7]. Given a logic program \( P \) and \( S \subseteq B_P \), we define \( \Gamma_P(S) = \text{least}(P^S) \) where \( P^S = \{ \text{head}(r) \leftarrow \text{body}^+(r) \mid r \in P, \text{body}^-(r) \cap S = \emptyset \} \) and \( \text{least}(P^S) \) is the least model of the positive logic program \( P^S \). The square of \( \Gamma_P \), \( \Gamma_P^2 \), is a monotonic operator and thus has both a least fixpoint, \( \text{lfp}(\Gamma_P^2) \), and a greatest fixpoint \( \text{gfp}(\Gamma_P^2) \). We obtain the well-founded model \( \text{WFM}(P) \) of a normal logic program \( P \) as \( \text{WFM}(P) = \text{lfp}(\Gamma_P^2) \cup \text{not } (\text{wfm}(B_P \setminus \text{gfp}(\Gamma_P^2))) \).

Two programs \( P \) and \( P' \) are equivalent (under WFS), denoted by \( P \equiv_{wf} P' \), iff \( \text{WFM}(P) = \text{WFM}(P') \). Finally, the inference relation under the WFS is defined for any literal \( q \in \text{Lit}(P) \) as follows: \( P \models_{wf} q \) iff \( q \in \text{WFM}(P) \).

### 3 Forgetting under the Well-Founded Semantics

When defining forgetting of an atom \( p \) in a given logic program \( P \), we want to obtain a new logic program \( P' \) such that it does not contain any occurrence of \( p \) or its default negation \( \text{not } p \). Additionally, we want to ensure that only the derivation for \( p \) (and \( \text{not } p \)) is affected, keeping \( P' \) and \( P \) equivalent w.r.t. all derivable literals excluding \( p \) (and \( \text{not } p \)). We want to achieve this based on the semantics rather than the syntax and ground it in forgetting in classical logic.

So, we semantically define the result of forgetting under the WFS by determining the well-founded model, and then providing a logic program that excludes \( p \) syntactically, and whose well-founded model excludes (only) \( p \) semantically.

**Definition 1.** Let \( P \) be a logic program and \( p \) an atom. The result of forgetting about \( p \) in \( P \), denoted \( \text{forget}(P,p) \), is a logic program \( P' \) such that the following two conditions are satisfied:

1. \( B_P \subseteq B_P \setminus \{ p \} \), i.e., \( p \) does not occur in \( P' \), and
2. \( \text{WFM}(P') = \text{WFM}(P) \setminus (\{ p \} \cup \{ \text{not } p \}) \)

This definition obviously does not introduce new symbols (cf. (F2) in [6]). In the rest of this section, we assume \( P, P' \) logic programs and \( p \) an atom, and show a number of desirable properties. The first one corresponds to (F3) in [6].

**Proposition 2.** For any \( l \in \text{Lit}(\{ p \} \cup \{ \text{not } p \}) \), \( \text{forget}(P,p) \models_{wf} l \) iff \( P \models_{wf} l \).

Our definition of forgetting also implies that there are syntactically different logic programs that correspond to \( \text{forget}(P,p) \). However, as we show next, all
Algorithm \text{forget}_1(P, p)

\textbf{Input}: Normal logic program \(P\) and an atom \(p\) in \(P\).
\textbf{Output}: A normal logic program \(P'\) representing \text{forget}(P, p).

\textbf{Method}:

\begin{enumerate}[Step 1.]
    \item Compute the well-founded model \(WFM(P)\) of \(P\).
    \item Let \(M\) be the three-valued interpretation obtained from \(WFM(P)\) by removing \(p\) and \(\text{not } p\). Construct a new logic program with \(B_{P'} = B_P \setminus \{p\}\) whose well-founded model is exactly \(M\):
        \[ P' = \{ a ← . \mid a \in M^+ \} \cup \{ a ← \text{not } a. \mid a \in B_{P'} \setminus (M^+ \cup M^-) \}. \]
    \item Output \(P'\) as \text{forget}(P, p).
\end{enumerate}

\textbf{Proposition 3.} If \(P'\) and \(P''\) are two results of \text{forget}(P, p), then \(P' \equiv_{wf} P''\).

Forgetting also preserves equivalence on \(\equiv_{wf}\) (cf. (F4) in [6]).

\textbf{Proposition 4.} If \(P \equiv_{wf} P'\), then \text{forget}(P, p) \equiv_{wf} \text{forget}(P', p).

However, our definition of forgetting preserves neither strong nor uniform equivalence. Intuitively, the reason is that Def. 1 only specifies the change on the semantics but not the precise syntactic form of the resulting program.

We may also generalize the definition of forgetting to a set of atoms \(S\) in the obvious way and show that the elements of the set can be forgotten one-by-one.

\textbf{Proposition 5.} Let \(P\) be a logic program and \(S = \{q_1, \ldots, q_n\}\) a set of atoms. Then \(\text{forget}(P, S) \equiv_{wf} \text{forget}(\text{forget}(P, q_1), \ldots, q_n)\).

We show that our notion of forgetting is faithful w.r.t. semantic forgetting in ASP [6] as follows, which also links to classical forgetting (cf. (F1) in [6]).

\textbf{Theorem 6.} Let \(P\) be a logic program and \(p, q\) atoms.
\begin{enumerate}
    \item If \(q \in WFM(\text{forget}(P, p))\), then \(q \in M\) for all \(M \in AS(\text{forget}_{ASP}(P, p))\).
    \item If \(\text{not } q \in WFM(\text{forget}(P, p))\), then \(q \notin M\) for all \(M \in AS(\text{forget}_{ASP}(P, p))\).
\end{enumerate}

\section{Computation of Forgetting}

\subsection{Naive Semantics-based Algorithm}

Def. 1 naturally leads to an algorithm for computing the result of forgetting about \(p\) in a given logic program \(P\): compute the well-founded model \(M\) of \(P\) and construct a logic program from scratch corresponding to \(WFM(\text{forget}(P, p))\). This idea is captured in Algorithm \text{forget}_1(P, p) shown in Fig. 1.
Algorithm forget₂(P, p)

Input: Normal logic program P and an atom p in P.
Output: A normal logic program P' representing forget(P, p).

Method:

Step 1. Query for the truth value of p in WFM(P) of P (e.g., using XSB).
Step 2. Remove all rules whose head is p. Moreover, given the obtained truth value of p in WFM(P), execute one of the three cases:

- t: Remove all rules that contain not p in the body, and remove p from all the remaining rule bodies.
- u: Substitute p and not p in each body of a rule r in P by not head(r).
- f: Remove all rules that contain p in the body, and remove not p from all the remaining rule bodies.

Step 3. Output the result P' as forget(P, p).

Fig. 2. Algorithm forget₂(P, p)

4.2 Query-based Algorithm

Algorithm forget₁(P, p) has two shortcomings. First, the syntactical structure of the original logic program is completely lost, which is not desirable if the rules are subject to later update or change: the author would be forced to begin from scratch, since the originally intended connections in the rules were lost in the process. Second, the computation is not particularly efficient, e.g., if we consider a huge number of rules from which we want to forget one atom p only.

In the following, we tackle the shortcomings of forget₁(P, p) based on the fact that the WFS is relevant, in the sense that it allows us to query for one atom in a top-down manner without having to compute the entire model.³ This means that we only consider a limited number of rules in which the query/goal or one of its subsequent subgoals appear. Once the truth value of p is determined, we only make minimal changes to accommodate the forgetting of p: if p is true (resp. false), then body atoms (resp. entire rules) are removed appropriately; if p is undefined, then all occurrences of p (and not p) are substituted by the default negation of the rule head, thus ensuring that the rule head will be undefined, unless it is true because of another rule in P whose body is true in WFM(P). The resulting algorithm forget₂(P, p) is shown in Fig. 2.

4.3 Forgetting as Program Transformations

What if we could actually avoid computing the well-founded-model at all? We investigate how to compute forget(P, p) using syntactic program transformations instead, thereby handling (F5) and completing the match to the criteria in [6].

³ See, e.g., XSB (http://xsb.sourceforge.net) for an implementation.
Algorithm \textit{forget}_3(P, p)

\textit{Input}: Normal logic program \(P\) and an atom \(p\) in \(P\).

\textit{Output}: A normal logic program \(P'\) representing \(\text{forget}(P, p)\).

\textit{Method}:

\textit{Step 1}. Compute \(\hat{P}\) by exhaustively applying the transformation rules in \(\mapsto_X\) to \(P\).

\textit{Step 2}. If neither \(p \leftarrow \cdot \in \hat{P}\) nor \(p \notin \text{heads}(\hat{P})\), then substitute \(p\) and \(\text{not } p\) in each body of a rule \(r\) in \(\hat{P}\) by \(\text{not } \text{head}(r)\). After that, remove all rules whose head is \(p\).

\textit{Step 3}. Output the result \(P'\) as \(\text{forget}(P, p)\).

\textbf{Fig. 3.} Algorithm \textit{forget}_3(P, p)

The basic idea builds on a set of program transformations \(\mapsto_X\) [3], which is a refinement of [2] for the WFS, avoiding the potential exponential size of the resulting program in [2] yielding the program remainder \(\hat{P}\). It is shown in [3] that \(\mapsto_X\) is always terminating and confluent and that the remainder resulting from applying these syntactic transformations to \(P\) relates to the well-founded model \(WFM(P)\) in the following way: \(p \in WFM(P)\) iff \(p \leftarrow \cdot \in \hat{P}\) and \(p \notin \text{heads}(\hat{P})\). We can use this to create the algorithm \textit{forget}_3(P, p) shown in Fig. 3 which syntactically computes the result of \(\text{forget}(P, p)\).

**Theorem 7.** Given logic program \(P\) and atom \(p\), \(\text{forget}_x(P, p)\), \(1 \leq x \leq 3\), computes a correct result of \(\text{forget}(P, p)\), terminates, and computing \(P'\) is in \(P\).

\section{Conclusions}

We have developed a notion of semantic forgetting under the well-founded semantics and presented three different algorithms for computing the result of such forgetting, and in each case the computational complexity is in \(P\).

In terms of future work, we intend to pursue different lines of investigation. First, we may consider a notion of forgetting that also preserves strong equivalence for different programs, similar to [15] for the answer set semantics, possibly based on HT\(^2\) [4] or adapting work on updates using SE-models [13,14]. An important issue then is whether the result is again expressible as a normal logic program. Second, since forgetting has been considered for description logics (DLs), we may also consider forgetting in formalisms that combine DLs and non-monotonic logic programming rules under WFS, such as [8].

\textbf{Acknowledgments} J. Alferes and M. Knorr were partially supported by FCT (Fundação para a Ciência e a Tecnologia) under project “ERRO – Efficient Reasoning with Rules and Ontologies” (PTDC/EIA-CCO/121823/2010), and M. Knorr also by FCT Grant SFRH/BPD/86970/2012. K. Wang was partially supported by Australian Research Council under grants DP110101042 and DP1093652.
References